EXHIBIT D

I hereby certify that this correspondence is being filed using EFS-Web addressed to: Mail Stop Inter Partes Reexam, Central Reexamination Unit, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on the date shown below.

Dated: March 1, 2011 Signature: / Robert T. Neufeld /
Patent Attorney & Reg. No. 48,394

Docket No. 13557.105125
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Reexamination of:	
Fresko	Control No.: Not Yet Assigned
Patent No.: 7,426,720	Examiner: Not Yet Assigned
Issue Date: September 16, 2008	Art Unit: Not Yet Assigned
For: SYSTEM AND METHOD FOR DYNAMIC PRELOADING OF CLASSES THROUGH MEMORY SPACE CLONING OF A MASTER RUNTIME SYSTEM PROCESS	

REQUEST FOR *INTER PARTES* REEXAMINATION UNDER 37 C.F.R. § 1.915

Mail Stop Inter Partes Reexam
Attn: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Google Inc. (hereinafter, "Requester") submits, under the provisions of 37 C.F.R. § 1.902 *et seq.*, a Request for Reexamination (hereinafter, "Request") of claims 1-8, 10-17, and 19-22 of U.S. Patent No. 7,426,720 (hereinafter "the '720 patent") entitled "System and Method for Dynamic Preloading of Classes through Memory Space Cloning of a Master Runtime System Process," issued to Fresko on Sept. 16, 2008. The '720 patent is provided as Exhibit 1 to the Request.

In support of its request, Requester provides the following:

   a. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S.C. § 103(a) as rendered obvious by Dike in view of Steinberg.

4. Obviousness under 35 U.S.C. § 103(a) based on the Bryant and APA-Bach references.

   a. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S.C. § 103(a) as rendered obvious by Bryant in view of APA-Bach.

5. Obviousness under 35 U.S.C. § 103(a) based on the Bryant and Traut references.

   a. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S.C. § 103(a) as rendered obvious by Bryant in view of Traut.

6. Obviousness under 35 U.S.C. § 103(a) based on the Srinivasan and APA-Bach references.

   a. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S.C. § 103(a) as rendered obvious by Srinivasan in view of APA-Bach.

7. Obviousness under 35 U.S.C. § 103(a) based on the Sexton and Bugnion references.

   a. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S.C. § 103(a) as rendered obvious by Sexton in view of Bugnion.

8. Obviousness under 35 U.S.C. § 103(a) based on the Sexton and Johnson references.

   a. Claims 1-8, 10-17, and 19-22 are unpatentable under 35 U.S.C. § 103(a) as rendered obvious by Sexton in view of Johnson.

H. Overview of Substantial New Questions of Patentability

<u>Webb in view of Kuck and further in view of APA-Bach</u>

The Examiner allowed the ‘720 patent to issue because, in his view, the combination of Webb and Kuck did not disclose the copy-on-write limitation that the Applicant added to each and every claim of the ‘720 patent. The APA-Bach reference, incorporated by reference in the ‘720 patent but neither included in an Information Disclosure Statement nor considered by the Examiner during prosecution, discloses the copy-on-write limitation and therefore poses a substantial new question of patentability.

Both Webb and Kuck are prior art to the '720 patent, consistent with the Examiner's findings during the prosecution of the '720 patent. *See* Non-Final Rejection at 3. The APA-Bach reference was published in 1986, and is prior art to the '720 patent under 35 U.S.C. § 102(b), given a priority date for the '720 patent of December 22, 2003. The APA-Bach reference was not considered by the Patent Office during the prosecution of the application that matured into the '720 patent, and therefore the combination of Webb, Kuck, and APA-Bach is not cumulative to the prior art considered by the Patent Office during the prosecution of the '720 patent.

As the Examiner discussed, Webb discloses a computer system including a memory and a processor. *See* Webb at 3:56-4:2. Webb implements a class pre-loader: "[a]t run-time objects are created as instantiations of these class files, and indeed the class files themselves are effectively loaded as objects." *See* Webb at 1:22–38. Webb's disclosure of a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process is clear in Fig. 3, which Webb explains is the "initialization of a loaded class (step 350), which represents calling the static initialization method (or methods) of the class . . . this application must be performed once and only once before the first active use of a class" which then allows for "[t]he new application class [to load] the application classes into the JVM (step 410), and involves the steps shown in Fig. 3 for all the application classes." *See* Webb at 6:59-65 & 7:34-39.

Next, the Examiner relied on Kuck to disclose a runtime environment to clone the memory space as a child runtime system process responsive to a process request and to execute the child runtime system process. *See* Kuck at ¶¶ [0064]-[0065]. Thus the Examiner found that all of the claim elements of independent Claim 1, which were identical to the issued claim 1 save

March 1, 2011

Respectfully submitted,

By / Robert T. Neufeld/
Robert T. Neufeld
Patent Attorney
Registration No. 48,394
KING & SPALDING LLP
1180 Peachtree Street
Atlanta, Georgia 30309

EXHIBIT 17: WEBB IN VIEW OF KUCK AND FURTHER IN VIEW OF APA-BACH

U.S. Patent No. 6,823,509
"Virtual Machine with Reinitialization"
Inventor: Alan Michael *Webb*
Assignee: International Business Machines Corporation
Filing Date: Dec. 20, 2000
Publication Date: Dec. 12, 2001
Issue Date: Nov. 23, 2004
("*Webb*")

U.S. Patent Publication No. 2003/0088604 A1
"Process Attachable Virtual Machines"
Inventors: Norbert *Kuck* et al.
Assignee: SAP AG
Filing Date: Nov. 7, 2002
Publication Date: May 8, 2003
Issue Date: Jan. 29, 2008 (as U.S. 7,325,233)
("*Kuck*")

M. J. Bach, The Design of the Unix Operating System, Bell Telephone Labs., Inc. (1986) ("*APA-Bach*")

The '720 patent issued from U.S. Patent Application Serial No. 10/745,023, filed on December 22, 2003. The United States Patent & Trademark Office issued a non-final rejection on April 27, 2007, then a final rejection on October 18, 2007. In both instances, all claims of the application were rejected as being unpatentable over the prior art of record, i.e., *Webb* and *Kuck*. The Examiner found that it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify *Webb's* teaching by adding a runtime environment to clone a parent memory space as a child runtime process responsive to a process request and to execute the child runtime process as taught by *Kuck*. See Non-Final Rejection (Apr. 27, 2007) at 5 (attached as Exhibit 4).

In response to these rejections, and in conjunction with the Applicant's filing of a Request for Continued Examination, the Applicant added the "copy-on-write" limitation to all independent claims, such that each and every claim of the '720 patent was amended to include—and presently contains—a "copy-on-write" limitation. *See* Amendment after Final Rejection (Dec. 18, 2007) (attached as Exhibit 5). Indeed, in their letter to, and interview with, the Examiner, the Applicant relied heavily on the "copy-on-write" cloning mechanism to distinguish their Application from the prior art, including *Webb* and *Kuck*, *see* Letter to Examiner Junchun Wu, which the Examiner acknowledged in the Notice of Allowability at 3.

Thus, it is clear from the record that the patentability of the '720 patent hinged on the purported novelty of the "copy-on-write" limitation. However, the application of a "copy-on-write" mechanism to the standard Unix or Linux fork( ) system call was well-known in the art prior to the filing of the application for the '720 patent, and was in fact explicitly disclosed in *APA-Bach*. Accordingly, as discussed in the chart below, particularly in light of the prosecution history, the Examiner's rejections and the Applicant's own statements, all of the claims of the '720 patent are unpatentable over the prior art of record in view of *APA-Bach*.

U.S. Patent No. 7,426,720	***Webb* and *Kuck* in view of *APA-Bach***
1. A system for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:	*Webb* and *Kuck* with *APA-Bach* provided a system for dynamic preloading of classes through memory space cloning of a master runtime system process. *Webb* discloses a system for preloading classes with a hierarchy of class loaders. *Webb* at col. 4, ll.26-41 ("JVM includes a hierarchy of class loaders"). *Kuck* discloses memory space cloning as a means to streamline and accelerate class loading: "Rather than initializing a new PAVM from scratch, the memory block of the master PAVM can simply be copied into the memory block of the new PAVM." *Kuck* at ¶¶ [0064]-[0065]. The Examiner cited *Webb* at col. 4, ll. 26-41 as disclosing a system directed to the dynamic preloading of classes through a cloning mechanism. *Non-Final Rejection*, Apr. 27, 2007, at 3. *Webb* explicitly discloses such a system for use in the implementation of Java Virtual Machines in numerous locations throughout the reference.
A processor; A memory	*Webb* and *Kuck* with *APA-Bach* provided a system that ran on a computer with a processor and memory. Each of the *Webb* and *Kuck* references explicitly disclose a system with a processor and memory, as shown below. "FIG. 1 illustrates a computer system 10 including a (micro)processor 20 which is used to run software loaded into memory 60. The software can be loaded into the memory by various means (not shown), for example from a removable storage device such as a floppy disc or CD ROM, or over a network such as a local area network (LAN) or

U.S. Patent No. 7,426,720	***Webb* and *Kuck* in view of *APA-Bach***
	telephone/modem connection, typically via a hard disk drive (also not shown). Computer system runs an operating system (OS) 30, on top of which is provided a Java virtual machine (JVM) 40. The JVM looks like an application to the (native) OS 30, but in fact functions itself as a virtual operating system, supporting Java application 50. A Java application may include multiple threads, illustrated by threads T 1 and T 2 71, 72." *Webb*, col. 3, l.56-col. 4, l.2. "[0029] Referring to FIG. 1, a network 10 includes a server 12 linked to client systems 14, 16, 18. The server 12 is a programmable data processing system suitable for implementing apparatus or performing methods in accordance with the invention. The server 12 contains a processor 16 and a memory 18. Memory 18 stores an operating system (OS) 20, a Transmission Control Protocol/Internet Protocol (TCP/IP) stack 22 for communicating over the network 10, and machine-executable instructions 24 executed by processor 16 to perform a process 500 below. In some implementations, the server 12 can contain multiple processors, each of which can be used to execute the machine-executable instructions 24." *Kuck*, ¶ [0029].
a class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code;	*Webb* and *Kuck* with *APA-Bach* provided a class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code. As quoted above, *Webb* clearly discloses a class preloader; the more detailed quotations below illustrate the substantive disclosure of *Webb*. "Java is an object-oriented language. Thus a Java program is formed from a set of class files having methods that represent sequences of instructions (somewhat akin to subroutines). A hierarchy of classes can be defined, with each class inheriting properties (including methods) from those classes which are above it in the hierarchy. For any given class in the hierarchy, its descendants (i.e. below it) are call subclasses, whilst its ancestors (i.e. above it) are called superclasses. At run-time objects are created as instantiations of these class files, and indeed the class files themselves are effectively loaded as objects. One Java object can call a method in another Java object. In recent years Java has become very popular, and is described in many books, for example "Exploring Java" by Niemeyer and Peck, O'Reilly & Associates, 1996, USA, and "The Java Virtual Machine Specification" by Lindholm and Yellin, Addison-Wedley, 1997, USA." *Webb*, col. 1, ll.22–38.

U.S. Patent No. 7,426,720	*Webb* and *Kuck* in view of *APA-Bach*
	"FIG. 2 shows the structure of JVM 40 in more detail (omitting some components which are not directly pertinent to an understanding of the present invention). The fundamental unit of a Java program is the class, and thus in order to run any application the JVM must first load the classes forming and required by that application. For this purpose the JVM includes a hierarchy of class loaders 110, which conventionally includes three particular class loaders, named Application 120, Extension 125, and Primordial 130. An application can add additional class loaders to the JVM (a class loader is itself effectively a Java program). In the preferred embodiment of the present invention, a fourth class loader is also supported, Middleware 124. Classes which are loaded by this class loader will be referred to hereinafter as middleware, whilst those loaded by Application Class loader 120 will be referred to as application." *Webb*, col. 4, ll.26–41. This is consistent with the Examiner's citation to *Webb* at col. 4, ll. 26-41, disclosing that "JVM includes a hierarchy of class loaders 110." *Non-Final Rejection*, Apr. 27, 2007, at 3.
a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process;	*Webb* and *Kuck* with *APA-Bach* provided a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process. The starting point for the *Webb* and *Kuck* with *APA-Bach* functionality is the instantiation of a master runtime process. This master runtime process will be cloned as further described below. The master runtime process is described in the *Webb* reference as application classes loaded into the Java Virtual Machine. *Webb* at col. 7, ll. 25-40. "The final step in FIG. 3 is the initialization of a loaded class (step 350), which represents calling the static initialization method (or methods) of the class. According to the formal JVM specification, this initialization must be performed once and only once before the first active use of a class, and includes things such as setting static (class) variables to their initial values (see the above-mentioned book by Lindholm and Yellin for a definition of "first active use"). Note that initialization of an object also requires initialization of its superclasses, and so this may involve recursion up a superclass tree in a similar manner to that described for resolution. The initialization flag in a class file 145 is set as part of the initialization process, thereby ensuring that the class is not subsequently re-initialised." *Webb*, col. 6, l.59 - col. 7, l.5.

U.S. Patent No. 7,426,720	*Webb* and *Kuck* in view of *APA-Bach*
	the class files themselves are effectively loaded as objects. One Java object can call a method in another Java object. In recent years Java has become very popular, and is described in many books, for example "Exploring Java" by Niemeyer and Peck, O'Reilly & Associates, 1996, USA, and "The Java Virtual Machine Specification" by Lindholm and Yellin, Addison-Wedley, 1997, USA." *Webb*, col. 1, ll.22–38. [0066] After a PAVM has been initialized, the PAVM can be used to process user requests from the corresponding user session. When a user request from the corresponding user session is received (504), an available process from the pool of OS processes allocated to the server can be selected to process the request (506). The PAVM of the user session that sent the request is then bound to the selected process (508)." *Kuck*, ¶ [0066]. The Examiner cited to *Webb*'s col. 4, ll. 27-29 disclosure that "[t]he fundamental unit of a Java program is the class, and thus in order to run any application the JVM must first load the classes forming and required by that application." *Non-Final Rejection*, Apr. 27, 2007, at 11.
10. A method for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:	*Webb* and *Kuck* with *APA-Bach* provided a method for dynamic preloading of classes through memory space cloning of a master runtime system process. The disclosures relevant to claim 1 are also relevant here. Specifically, *Webb* discloses a system for preloading classes with a hierarchy of class loaders. *Webb* at col. 4, ll.26-41 ("JVM includes a hierarchy of class loaders"). And *Kuck* discloses memory space cloning as a means to streamline and accelerate class loading: "Rather than initializing a new PAVM from scratch, the memory block of the master PAVM can simply be copied into the memory block of the new PAVM." *Kuck* at ¶¶ [0064]-[0065]. The Examiner cited *Webb* at col. 4, ll. 26-41 as disclosing a system directed to the dynamic preloading of classes through a cloning mechanism. *Non-Final Rejection*, Apr. 27, 2007, at 3. *Webb* explicitly discloses such a system for use in the implementation of Java Virtual Machines in numerous locations throughout the reference.
executing a master runtime system process;	*Webb* and *Kuck* with *APA-Bach* provided a method for executing a master runtime system process.

U.S. Patent No. 7,426,720	*Webb* and *Kuck* in view of *APA-Bach*
	"The final step in FIG. 3 is the initialization of a loaded class (step 350), which represents calling the static initialization method (or methods) of the class. According to the formal JVM specification, this initialization must be performed once and only once before the first active use of a class, and includes things such as setting static (class) variables to their initial values (see the above-mentioned book by Lindholm and Yellin for a definition of "first active use"). Note that initialization of an object also requires initialization of its superclasses, and so this may involve recursion up a superclass tree in a similar manner to that described for resolution. The initialization flag in a class file 145 is set as part of the initialization process, thereby ensuring that the class is not subsequently re-initialised." *Webb*, col. 6, l.59 - col. 7, l.5. "FIG. 4 illustrates a method in accordance with the present invention whereby this problem can be largely overcome. The method starts with the middleware initiating an application (e.g. a transaction), it being assumed that both the relevant part of the middleware and the application here are Java programs. In order to do this, the middleware creates an instance of the application class loader (it cannot use an existing one because the application class loader is below the middleware class loader in the class loader hierarchy as shown in FIG. 2, and so it has no reference to it). The new application class loader instance then loads the application classes into the JVM (step 410), and involves the steps shown in FIG. 3 for all the application classes. The application is then run in standard fashion (step 420), and after completion control returns to the middleware." *Webb*, col. 7, ll.25-40. This is consistent with the Examiner's citation to the col. 6, ll. 59-65 disclosure of "Fig. 3 is the initialization of a loaded class (step 350), which represents calling the static initialization method (or methods) of the class . . . the formal JVM specification, this initialization must be performed once and only once before the first active use of a class." *Non-Final Rejection*, Apr. 27, 2007, at 4.
obtaining a representation of at least one class from a source definition provided as object-oriented program code;	*Webb* and *Kuck* with *APA-Bach* provided a method for obtaining a representation of at least one class from a source definition provided as object-oriented program code. "Java is an object-oriented language. Thus a Java program is formed from a set of class files having methods that represent sequences of instructions (somewhat akin to subroutines). A

U.S. Patent No. 7,426,720	***Webb* and *Kuck* in view of *APA-Bach***
	hierarchy of classes can be defined, with each class inheriting properties (including methods) from those classes which are above it in the hierarchy. For any given class in the hierarchy, its descendants (i.e. below it) are call subclasses, whilst its ancestors (i.e. above it) are called superclasses. At run-time objects are created as instantiations of these class files, and indeed the class files themselves are effectively loaded as objects. One Java object can call a method in another Java object. In recent years Java has become very popular, and is described in many books, for example "Exploring Java" by Niemeyer and Peck, O'Reilly & Associates, 1996, USA, and "The Java Virtual Machine Specification" by Lindholm and Yellin, Addison-Wedley, 1997, USA." *Webb*, col. 1, ll.22–38. "FIG. 2 shows the structure of JVM 40 in more detail (omitting some components which are not directly pertinent to an understanding of the present invention). The fundamental unit of a Java program is the class, and thus in order to run any application the JVM must first load the classes forming and required by that application. For this purpose the JVM includes a hierarchy of class loaders 110, which conventionally includes three particular class loaders, named Application 120, Extension 125, and Primordial 130. An application can add additional class loaders to the JVM (a class loader is itself effectively a Java program). In the preferred embodiment of the present invention, a fourth class loader is also supported, Middleware 124. Classes which are loaded by this class loader will be referred to hereinafter as middleware, whilst those loaded by Application Class loader 120 will be referred to as application." *Webb*, col. 4, ll.25-41. For a similar type of rejection, the Examiner cited to *Webb*'s col. 4, ll. 27-29 disclosure that "[t]he fundamental unit of a Java program is the class, and thus in order to run any application the JVM must first load the classes forming and required by that application." *Non-Final Rejection*, Apr. 27, 2007, at 11.
interpreting and instantiating the representation as a class definition in a memory space of the master runtime system process;	*Webb* and *Kuck* with *APA-Bach* provided a method for interpreting and instantiating the representation as a class definition in a memory space of the master runtime system process. "The final step in FIG. 3 is the initialization of a loaded class (step 350), which represents calling the static initialization method (or methods) of the class. According to the formal JVM specification, this initialization must be performed once and only once before the first